Solving the compressible Navier-Stokes equations on up to 1.97 million cores and 4.1 trillion grid points

Iván Bermejo-Moreno
Center for Turbulence Research
Stanford University
Stanford, CA 94305-3024
ibermejo@stanford.edu

Julien Bodart
Center for Turbulence Research
Stanford University
Stanford, CA 94305-3024
jbodart@stanford.edu

Johan Larsson
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742
jola@umd.edu

Blaise M. Barney
Lawrence Livermore National Laboratory
Livermore, CA 94550
blaiseb@llnl.gov

Joseph W. Nichols
Center for Turbulence Research
Stanford University
Stanford, CA 94305-3024
jwn@stanford.edu

Steve Jones
HPC Center
Stanford University
Stanford, CA 94305-3024
stevejones@stanford.edu

ABSTRACT

We present weak and strong scaling studies as well as performance analyses of the *Hybrid* code, a finite-difference solver of the compressible Navier-Stokes equations on structured grids used for the direct numerical simulation of isotropic turbulence and its interaction with shock waves. Parallelization is achieved through MPI, emphasizing the use of non-blocking communication with concurrent computation. The simulations, scaling and performance studies were done on the *Sequoia*, *Vulcan* and *Vesta* Blue Gene/Q systems, the first two accounting for a total of 1,966,080 cores when used in combination. The maximum number of grid points simulated was 4.12 trillion, with a memory usage of approximately 1.6 PB. We discuss the use of hyperthreading, which significantly improves the parallel performance of the code on this architecture.

Categories and Subject Descriptors

I.6.3 [**Simulation and modeling**]: Applications

Keywords

Compressible turbulence, high-performance computing, Direct Numerical Simulation, Navier-Stokes, shock waves

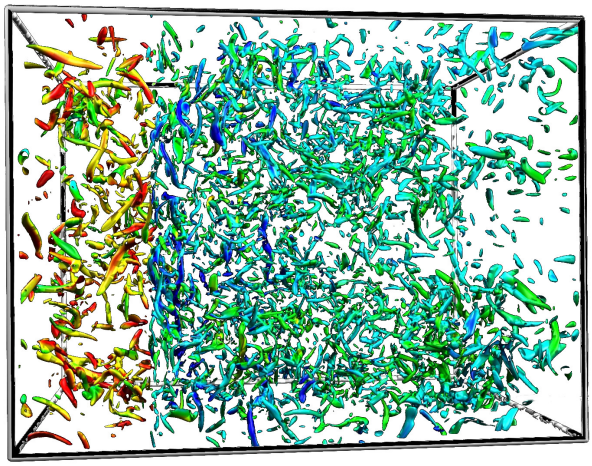


Figure 1: Visualization of turbulence eddies, flowing from left to right, modified by a shock wave nominally perpendicular to the incoming flow. Eddies are colored by the streamwise velocity changing abruptly from high (in red and yellow) to low (in blue and cyan) as they traverse the shock.

1. INTRODUCTION

The interaction of a shock wave with compressible turbulence is a canonical physical phenomenon that occurs in a wide range of scientific and engineering applications. Examples include high-speed flight and propulsion, space re-entry, inertial confinement fusion, but also the removal of kidney stones through lithotripsy. In all of these applications, the shock wave intensifies and/or leads to the generation of turbulence that considerably affects the performance of the device in question. A visualization of the shock/turbulence interaction phenomenon is shown in figure 1. The difficulty of performing experiments that cleanly isolate the phenomena involved in canonical shock/turbulence interaction has rendered direct numerical simulation (DNS), in which the continuum equations are solved without modeling, as a scientific tool of paramount importance to advance our understanding of the physics behind this problem.

The key parameter in turbulence simulations is the Reynolds number, Re, which is the ratio between inertial and viscous forces. As Re increases, the ratio of the length scales at

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SC13, November 17-21 2013, Denver, CO, USA

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-2378-9/13/11...$15.00.
http://dx.doi.org/10.1145/2503210.2503265

which the most energy resides (L) to those at which the most dissipation of turbulence energy into heat occurs (η) increases (see figure 2). At very high Re, which occurs in the majority of fluid flows found in nature and in engineering applications, this ratio L/η is sufficiently large to create a gap between the energy and dissipation spectra; this is clearly not the case in figure 2, and the main driver in fundamental turbulence simulations during the last decades has been to reach sufficiently high Re to be able to study realistic turbulence. DNS of isotropic homogeneous turbulence have reached values of the Reynolds number based on the Taylor microscale, Re_λ, in the order of 1,000 [7]. For simulations of shock/turbulence interaction, the Re_λ of the incoming turbulence upstream of the shock wave is increased by the compression experienced as the turbulence traverses the shock. Thus, the grid resolution required to resolve the turbulence scales downstream of the interaction imposes a large constraint on the Re_λ of the incoming turbulence that can be achieved in a DNS. The solid blue curve in figure 2 represents the current state-of-the-art for the canonical problem of shock/turbulence interaction, which corresponds to a value of $Re_\lambda = 75$ for the incoming turbulence. The spectra at that highest Reynolds number in figure 2 then suggest that the ratio L/η needs to be increased by at least a factor of 4 in order to have a "spectral gap" between the energetic and dissipative scales of turbulence. The number of grid points required in each spatial direction is directly proportional to this ratio L/η, as is the required number of time steps. Therefore, such a calculation would be 4^4=256 times more expensive than the current state-of-the-art for the canonical shock/turbulence interaction problem. An estimate of the time-to-solution for such a calculation is given in section 6.

The present work started in late 2007 under the Department of Energy SciDAC-2 program, with the aim of performing the most highly resolved simulations of canonical shock/turbulence interaction to date. Prior to this work, the largest simulations in the literature were conducted with grids up to $231\times81^2 = 1.5\cdot10^6$ grid points in the directions normal and tangential to the shock, respectively [11, 12]. In phase 1 of the present project, a solution-adaptive numerical method that accurately captures shock waves while preserving the broadband chaotic nature of turbulence was developed and implemented in the *Hybrid* code. In phase 2, this code was used to compute a set of 14 simulations on grids with $1{,}257\times384^2 = 190\cdot10^6$ [10]. In phase 3, four additional cases on finer grids with $2{,}366\times1{,}024^2 = 2.5\cdot10^9$ points were computed [9]. Note that the data in figure 2 was taken from the pre-shock turbulence in these two studies.

In the present phase of this work, early access to the Blue Gene/Q *Sequoia* and *Vulcan* clusters at the Lawrence Livermore National Laboratory was used to assess the scaling performance of the code and compute shock/turbulence interaction at unprecedented scale.

This manuscript is structured as follows: section 2 describes the current state-of-the-art regarding numerical solvers of related flows. Section 3 introduces the equations to be solved followed by the discretization, numerical algorithms and parallelization techniques used in the *Hybrid* code. In section 4 we describe the systems where the numerical simulations and scaling/performance tests were run. Section 5 introduces the applications used to measure performance and scalability. Results are presented and discussed in section 6.

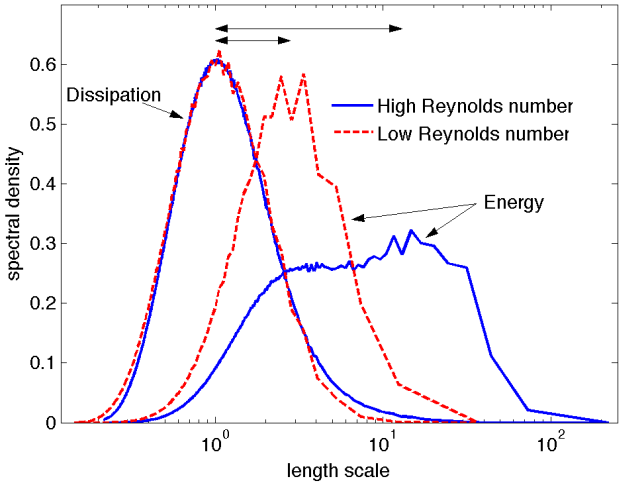


Figure 2: Spectral density of energy (rightmost curves) and the rate at which it dissipates into heat (leftmost curves, collapsed onto each other) for canonical isotropic turbulence. Results shown at increasing Reynolds numbers ($Re_\lambda = 40$ and 75, respectively, where Re_λ is the Reynolds number based on the Taylor microscale), computed using grids of 256^3 and $1{,}024^3$ grid points. The ratio of the most energetic length scale (L) to the one where the dissipation peaks (η) is highlighted with double-arrows; note how this ratio L/η increases with the Reynolds number.

2. CURRENT STATE-OF-THE-ART

The state-of-the-art of numerical methods for direct numerical simulation (DNS) of canonical turbulence problems depends on the nature of the turbulence being studied. The most fundamental case is that of homogeneous, isotropic, incompressible turbulence (HIIT). This is studied in three-dimensional periodic boxes, and thus spectral methods based on Fourier series expansions have been popular since the very first attempts at DNS [17].

Spectral methods were shown to be very efficient on vector supercomputers by Ishihara *et al.* [7] where $4{,}096^3$ Fourier modes were used to simulate HIIT. Note that time integration had to be done in single precision to satisfy memory requirements. They reported that 16.4 TFLOPS were sustained for a smaller grid size of $2{,}048^3$. On scalar architectures, similar resolution was achieved for a few time steps in the work of Donzis *et al.* [3]. To efficiently compute an FFT in a domain of size N_p^3 with $N_p = O(10^3)$ using $N_c = O(10^4)$ cores, the slice decomposition had to be extended to linelet decomposition and the *p3dfft* library was developed. In this framework nearly perfect scaling was achieved, provided that the number of grid points per core is kept sufficiently high to hide the expensive transposition to the physical space required by pseudo-spectral algorithms. Linear scaling of the FFT itself was achieved up to 65K cores by Pekurovsky [14]. The high efficiency of spectral methods has also been shown in the context of wall-bounded turbulence, another canonical problem of interest to the fluid mechanics community. Schlatter and Orlu [18] used such a method with $8{,}192\times513\times768 = 3.2\cdot10^9$ modes to com-

pute the turbulent flow in a flat-plate boundary layer. However, to maintain a roughly constant time-to-solution for ever larger problem sizes, the amount of work done per core must decrease since the number of time steps required is proportional to the number of spatial modes. Scaling FFT algorithms, specifically the transpose operations, in this context remains a challenge.

Alternatively, compact-stencil based methods like the finite difference method with high (>2) order of accuracy are attractive and are likely to become the new standard for direct numerical simulation. The small sacrifice in accuracy is compensated for by better parallel performance due to the lower communication requirements, with only neighbor-to-neighbor type communications. Pirozzoli and Bernardini [16] performed the largest simulation of this kind to date, computing a flat-plate boundary layer using $16{,}384 \times 900 \times 2{,}304 = 34{\cdot}10^9$ grid points and a compressible flow solver.

A drawback of stencil-based methods is their low ($O(1)$) arithmetic intensity (AI) leading to low peak performance (in terms of the rate of floating point operations). In the context of turbulence in compressible flows, recent work by Hejazialhosseini *et al.* [6] employed the roofline model [19] to accomplish a high level of optimization, including NUMA-aware techniques and data re-ordering to increase cache locality. In some conditions the achieved performance was measured as high as 30% of the peak on a 47K-core cluster. In most of these prior studies the most important quantity of interest to the community, the time-to-solution, is rarely discussed (with a few exceptions as [3] for a spectral algorithm and [7] for the vector architectures they used). One can define the normalized time

$$t_w = \Delta^{wt} N_c / N_p \tag{1}$$

in which Δ^{wt} is the wall clock time needed to perform one time step, N_c and N_p being the number of cores used and the number of grid points (or control volumes), respectively Despite being algorithm-dependent (for example, varying between spectral methods with $AI \sim O(\log N)$ and stencil based methods with $AI \sim O(1)$), t_w is the only relevant metric when trying to solve a problem of a given size on a given machine. We provide t_w values found in our work, for benchmarking purposes, in section 6.

For the specific problem of shock/turbulence interaction, the largest computations to date were conducted during the first phases of the present study [10, 9]; specifically calculations using the *Hybrid* code on $2{,}366{\times}1{,}024^2 = 2.5{\cdot}10^9$ grid points using 64K cores of the Blue Gene/P *Intrepid* machine at the Argonne Leadership Computing Facility.

3. THE HYBRID CODE

The *Hybrid* code solves the compressible Navier-Stokes equations for a perfect gas on structured, Cartesian, stretched grids

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j) = 0, \tag{2}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_i u_j) = \frac{\partial \sigma_{ij}}{\partial x_j}, \tag{3}$$

$$\frac{\partial \rho e_T}{\partial t} + \frac{\partial}{\partial x_j}(\rho e_T u_j) = \frac{\partial u_i \sigma_{ij}}{\partial x_j} - \frac{\partial q_j}{\partial x_j}, \tag{4}$$

where the Einstein summation convention is implied; ρ is the density, u_i is the velocity component in the i-th coordinate direction ($i = 1, 2, 3$), $e_T = e + \frac{1}{2} u_i u_i$ is the total energy per unit mass (e is the internal energy per unit mass), q_j is the heat flux given by Fourier's law as $q_j = -\kappa\, \partial T / \partial x_j$ (T is the temperature and $\kappa(T)$ is the thermal conductivity), and σ_{ij} is the stress tensor, defined as:

$$\sigma_{ij} = -p\delta_{ij} + 2\mu \left(S_{ij} - \frac{\delta_{ij}}{3} \frac{\partial u_k}{\partial x_k} \right), \tag{5}$$

where p is the pressure, related to the other thermodynamic variables by the equation-of-state for a perfect gas $p = \rho R T$, R is the gas constant. μ is the dynamic viscosity, assumed to follow the power-law $\mu = \mu_{\text{ref}} (T/T_{\text{ref}})^{3/4}$. The convective term in equation 3, $\partial \rho u_i u_j / \partial x_j$, makes this system of five partial differential equations (PDEs) highly nonlinear, and is responsible for features like shock waves and broadband turbulence.

From a numerical point-of-view, the incompletely hyperbolic nature of the PDEs implies that explicit time integrators (most popularly, explicit Runge-Kutta methods) are often the most efficient, since small time steps are required to retain high accuracy anyway. On the other hand, the compressible Navier-Stokes equations are notoriously affected by nonlinear aliasing errors, through which numerical noise gets amplified and aliased to larger scales. This sensitivity to aliasing errors is most visible in the popularity of using numerical dissipation in different forms (upwinding, filters after each time step, explicitly added dissipative terms, essentially non-oscillatory schemes, etc.) to ensure a stable solution. However, this added numerical dissipation can be rather harmful to the accuracy of the results, given the prime importance of energy transfer mechanisms in the physics of turbulence.

One key aspect of the *Hybrid* code is the solution-adaptive nature of the algorithm. Adams and Shariff [1] introduced the idea of using different numerical schemes around a shock wave and in the rest of the domain. Specifically, they used an upwinded Pade-derivative in most of the domain coupled to an ENO scheme near shocks. Pirozzoli [15] later improved this method to make the coupling between the schemes conserve mass, momentum and energy. The implementation in the *Hybrid* code advances this one step further, by utilizing a split form of the convective terms that reduces the aliasing errors; this allows for stable computations with very little numerical dissipation. For example, in the present simulations, the physical viscosity is sufficient to maintain stability. We note that Pantano *et al.* [13] independently arrived at a similar split form.

The equations 2-4 are solved for the five conserved variables (ρ, ρu_i, ρe_T) in the *Hybrid* code. The computational domain is decomposed at initialization into blocks according to the number of MPI tasks, aiming to ensure load balance and minimize the MPI communication time by reducing the number of grid points in the block interfaces. For runs with a homogeneous cost per grid point, blocks with the closest number of points among coordinate directions, ideally cubic, are sought. In each stage of the time stepping algorithm, the contribution of the convective terms (inviscid fluxes) is first calculated, followed by the computation of viscous fluxes. Primitive variables (u, p, T) as well as velocity gradients ($\partial u_i / \partial x_j$), are then computed from the updated conserved quantities. A sensor quantity, based on the enstrophy and

dilatation obtained from the velocity gradients, is then calculated at each grid point to mark regions near shock waves (discontinuities) where the WENO scheme with Roe flux splitting will be used for the approximation of the inviscid fluxes in the subsequent time-stepping stage.

The most common configuration of the *Hybrid* code is to use a 6th-order accurate central-difference scheme in most of the domain, and a 5th-order accurate WENO scheme across all discontinuities, marked by the sensor. The accuracy of these schemes to solve the physics of shocks and turbulence was thoroughly assessed in [8]. Time integration is done using a standard 4th-order accurate, 4-stage Runge-Kutta scheme. With these choices, specifically the use of a spatially explicit difference operator, the parallel communication is minimized. The code is written in C++ and parallelized using MPI, including all major I/O. It uses double-precision arithmetic, with a memory footprint of 42 doubles per grid point in every block, thus including ghost (halo) points. All exchanges of halo-data are done through non-blocking and asynchronous communication with concurrent computation to hide the communication cost. For example, the computation of primitive variables from the conserved ones (which are updated in the Runge-Kutta time-stepping) is done by first computing primitive variables in all interior grid points of the block (i.e., within 3 grid points of the block boundaries, for the order of the numerical schemes under consideration), then initiating the communication of both conserved and primitive variables (through a non-blocking `MPI_Isend`/`MPI_Irecv` pair), and computing primitive variables in all remaining grid points near the block boundaries before the communications are terminated. Similarly, communication of the velocity gradients is started at the beginning of each stage of the Runge-Kutta time-stepping, before the calculation of inviscid fluxes and ended prior to the computation of viscous fluxes, for which they are required.

4. HARDWARE AND ENVIRONMENT

In this study, the performance and scalability measurements of the *Hybrid* code were done on IBM Blue Gene/Q platforms, including *Sequoia*, *Vulcan* and *Vesta* clusters. The first two are located at the Lawrence Livermore National Laboratory, whereas the latter is located at the Argonne National Laboratory. The *Hybrid* code has been previously tested on multiple platforms, including Cray XT-4, Blue Gene/P, Intel Westmere Rocks and AMD Opteron clusters.

Blue Gene/Q (BG/Q) is the third generation Blue Gene architecture developed by IBM, succeeding Blue Gene/L and Blue Gene/P. As its predecessors, BG/Q is based on a system-on-a-chip (SoC) building block [5]. Each SoC consists of eighteen 64-bit PowerPC A2 processor cores, sixteen of which are dedicated to computing. Of the other two processors, one serves the operating system and the other is reserved for redundancy. Two execution units, namely XU and AXU, are available per core. XU is capable of integer, load and store operations, whereas AXU is capable of floating-point operations. One instruction can be completed at most per cycle and thread. Two instructions, one from each of these two execution units, can be completed per cycle per core. Each core runs at 1.6 GHz and includes a SIMD quad-vector double precision floating point unit (QPX) that results in 4-way simultaneous hardware hyperthreading, bringing the peak performance per core to 12.8 GFLOPS. Each core also includes a 16 KiB L1 D-cache, with a 4 KiB prefetch buffer (L1P). The latencies for the L1 cache and the prefetch buffer are 6 and 24 cycles, respectively [2].

In addition to the processors, each SoC also integrates a 32 MiB eDRAM L2 cache, running at 800 MHz and connecting all cores with a full crossbar switch. The L2 latency is 84 cycles. Networking logic is also included at the chip level, for communications with other chips in a 5D torus configuration, through 2 GB/s chip-to-chip links. Each compute card (node) includes a chip mounted with 16 GiB of DDR3 DRAM (1 GiB per core). The latency for this off-chip memory is 346 cycles. Compute cards are then assembled onto a water-cooled node board (Q32) consisting of 32 nodes. A compute rack can contain up to 32 node boards (organized in two mid-planes of 16 node boards each), thus including 1,024 compute nodes, 16,384 cores and 16 TiB RAM.

Regarding the performance of memory-bound applications, such as the Hybrid code, a key aspect of the BG/Q architecture is the high latency for the L2 cache (84 cycles) in comparison with the nearly 10 cycles found in other popular CPUs. Hiding this latency can be partially achieved by increasing the number of instructions issued per cycle through hyperthreading, which allows multiple instruction streams (threads) to execute simultaneously per core. Thus, when one of the threads is idle (for example, waiting for data to be retrieved from cache/memory), the other thread(s) can still be executing instructions (at the maximum rate of two instructions per cycle, one per execution unit, in each PowerPC A2 core of BG/Q, which allows up to four threads).

The *Sequoia* cluster contains 96 racks, resulting in 1,572,864 cores, whereas the *Vulcan* cluster includes 24 racks (393,216 cores) and the *Vesta* cluster, intended for development and testing, has 2 racks (32,768 cores). The present study was done during an early-access phase scheduled at the Lawrence Livermore National Laboratory for the beginning of 2013. During this phase, the *Sequoia* and *Vulcan* clusters were occasionally inter-connected to perform scalability tests on a 120-rack configuration, resulting in an unprecedented core count of 1,966,080, and a combined theoretical peak performance of 25 PFLOPS.

The operating system installed on the compute nodes is the lightweight Compute Node Kernel (CNK). Dedicated IO nodes, running Red Hat Enterprise Linux, handle the file system input/output operations (in a proportion of one IO node per 128 compute nodes). The IBM XL compilers were used for all the tests reported in this work.

5. SCALABILITY/PERFORMANCE TOOLS

Several tools for measuring scalability and performance are combined in this work. First, the code is instrumented with MPI timing functions that measure the elapsed wall-clock time spent in different parts of the program execution, categorized into computation, MPI communication (including waiting times for send and receive operations, as well as barrier and reduce function calls) and IO. In this work, we focus on the global times spent in computation and communication, as they are the most dominant in the scientific simulations under consideration. For each simulation run, the timing information corresponds to averages over the number of time steps of execution. The resolution of the `MPI_Wtime` function in the BG/Q system, as returned by the `MPI_Wtick` function call, is one nanosecond, several orders of magnitude smaller than the times being measured. The overhead im-

posed by the instrumentation with MPI timers is minimal. Therefore this approach will be used to obtain scaling and time-to-solution information.

To measure performance, the IBM High Performance Computing Toolkit (IHPCT) is utilized. It provides MPI timing and profiling information, as well as data retrieved from hardware counters, through the Hardware Performance Monitor (HPM) component library. HPM uses the native Blue Gene Hardware Performance Monitoring (BGPM) API [4] to access subsets of the 424 64-bits hardware counters available in BG/Q from five different units: CPU, L2, I/O, network and CNK. Derived performance measurements, such as floating-point operations per second (FLOPS), are also provided by the HPM library.

6. RESULTS

In this section we present first weak and strong scaling results, followed by performance measurements and profiling data, identifying the bottlenecks that are generalizable to other time-explicit, structured-mesh CFD codes. We particularize for the two types of numerical schemes that are combined in the solution-adaptive methodology described in section 3: the 6th-order central-difference and the 5th-order WENO schemes. Periodic boundary conditions were used and no shocks are present in the flow to ensure stability when using only central-difference scheme. The computational cost per grid point associated with each one of these numerical schemes is rather different, whereas the communication load will be constant. When combined with the use of non-blocking MPI communications and the hyperthreading capabilities of the Blue Gene/Q architecture, this will result in differences in performance and scaling behavior that can be useful for the determination of load-balancing strategies.

6.1 Weak scaling

6.1.1 *Maximum memory load, no hyperthreading*

In this first weak scaling test we aim to maximize the memory usage. As explained in section 4, each node card has 16 GiB of memory, resulting in 1GiB per core, nominally. However, a fraction of the total memory will be used by the operating system (e.g., CNK consumes a fixed size of 16 MiB), program text, shared memory, MPI, etc. This fraction generally increases with the core count. Also, the use of multiple threads per core introduces an overhead that decreases the available total memory for the computation. Therefore, no hyperthreading is used in these first weak scaling runs. The block size chosen for this weak scaling test is 128^3 points per core. The memory footprint of the *Hybrid* code for such a block size is approximately 830 MB. Keeping this computational load per core constant, the number of cores is increased in successive runs from 32 to the 1,966,080 available when the *Sequoia* and *Vulcan* clusters were interconnected. 32 cores correspond to the minimal core count involving inter-node communications.

Table 1 shows the 17 different cases run for this weak scaling study with their corresponding number of cores, grid points per coordinate direction, total grid size, corresponding MPI block decomposition. All cases are run with a 6th-order central-difference numerical scheme. The required MPI communications are minimal when the differences between the number of points in each direction are smallest. We note that no optimized mappings of the physical torus network to the structured domain decomposition were utilized in these runs. The largest case, run on 120 racks, includes over 4.1 trillion points (20.5 trillion degrees of freedom), and a memory footprint surpassing 1.6 PB.

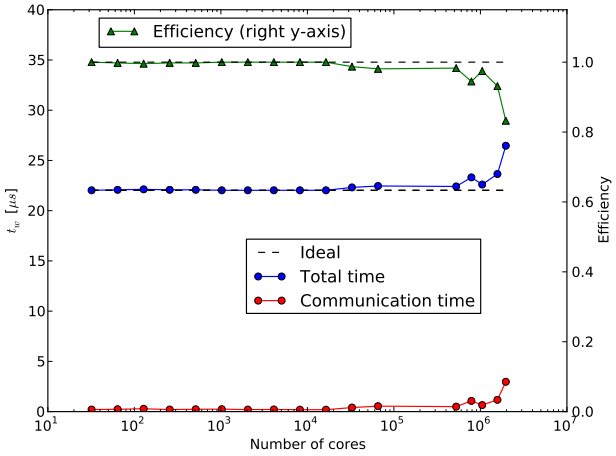


Figure 3: Weak scaling results on the combined *Sequoia* and *Vulcan* BG/Q clusters, for a unit block size of 128^3 grid points per core, increasing the core-count from 32 to 1,966,080. See table 1 for further details on each run.

Figure 3 shows the weak scaling results corresponding to table 1. The horizontal axis represents the number of cores, in logarithmic scale. The left vertical axis represents the metric, t_w, defined in equation 1, in microseconds. Besides the total wall-clock time per time step, plotted in blue, the figure also shows the communication time for each case, in red. The reported times have been averaged over the last five time steps of each run. Tests were also performed with 50 time steps per run for comparison, finding a difference of less than 1% in the results. The weak scaling efficiency is plotted in green and its magnitude is read on the right vertical axis.

It is observed that the weak scaling is nearly perfect up to 16,384 cores (1 rack). From 16,384 to 524,288 (32 racks) the percentage of time spent in MPI communications increases, and the efficiency decreases to 0.98. At 1,048,576 cores (48 racks) the efficiency drops to 0.94, recovering to 0.97 for 64 racks. This drop in efficiency was replicated in different tests performed at that scale and is probably resulting from the suboptimal communications associated with the block decomposition and MPI mapping layout. For the full *Sequoia* cluster (96 racks and 1,572,864 cores) the efficiency is 0.93, whereas adding the 24 additional racks of *Vulcan* brings the efficiency to 0.83. We note that the interconnection of both clusters was experimental: normal operation of the clusters during the early-access period in which these runs were performed, as well as the planned production runs, did not include the connection of *Sequoia* and *Vulcan*. Nevertheless, repeatability of all these runs was ensured by running on different occasions for which the two clusters were interconnected. The block decomposition resulting for this last case and the lack of an optimized MPI mapping layout to the torus network are also likely contributing to that drop in

Cores	N_x	N_y	N_z	N	Blocks	Efficiency
32	256	512	512	67,108,864	2×4×4	1.00
64	512	512	512	134,217,728	4×4×4	1.00
128	512	512	1,024	268,435,456	4×4×8	1.00
256	512	1,024	1,024	536,870,912	4×8×8	1.00
512	1,024	1,024	1,024	1,073,741,824	8×8×8	1.00
1,024	1,024	1,024	2,048	2,147,483,648	8×8×16	1.00
2,048	1,024	2,048	2,048	4,294,967,296	8×16×16	1.00
4,096	2,048	2,048	2,048	8,589,934,592	16×16×16	1.00
8,192	2,048	2,048	4,096	17,179,869,184	16×16×32	1.00
16,384	2,048	4,096	4,096	34,359,738,368	16×32×32	1.00
32,768	4,096	4,096	4,096	68,719,476,736	32×32×32	0.99
65,536	4,096	4,096	8,192	137,438,953,472	32×32×64	0.98
524,288	8,192	8,192	16,384	1,099,511,627,776	64×64×128	0.98
786,432	8,192	12,288	16,384	1,649,267,441,664	64×128×96	0.94
1,048,576	8,192	16,384	16,384	2,199,023,255,552	64×128×128	0.97
1,572,864	12,288	16,384	16,384	3,298,534,883,328	128×128×96	0.93
1,966,080	15,360	16,384	16,384	4,123,168,604,160	128×96×160	0.83

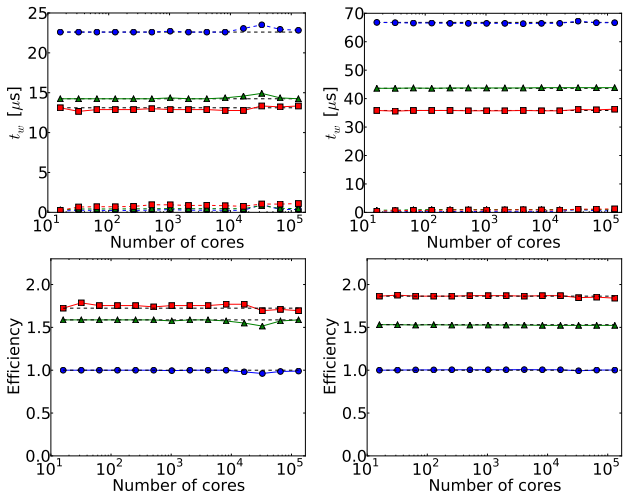
Table 1: Runs performed for the weak scaling study on *Sequoia* and *Vulcan* with the associated core count. grid dimensions, block decomposition and efficiency.

scaling efficiency.

6.1.2 *Hyperthreading and numerical scheme*

The next weak scaling tests are meant to evaluate the effects of hyperthreading on each numerical scheme (6th-order central-difference and 5th-order WENO). The use of hyperthreading in these runs required us to lower the memory footprint of the code by reducing the unit block size to 96^3 grid points per core. The number of cores was varied from 32 to 131,072.

Results are shown in figure 4: the top plots represent the wall-time (in μs) employed per number of grid points per core, t_w, and the bottom plots represent the weak scaling efficiency, for 6th-order central-difference (left column) and 5th-order WENO scheme (right column). Comparing the left and right top plots, it is noticed that WENO takes nearly three times longer on average than central-difference. The time spent in communications is nearly the same for both schemes, since the number of ghost points is kept constant in all runs. This results in an improved parallel scaling for WENO. It is also immediately seen that hyperthreading provides a substantial improvement of the efficiency, resulting in speedups larger than 1.5, sustained when the number of cores is increased. The gain is more noticeable when changing from a single task per core (no hyperthreading) to two. Further increasing from two tasks per core to four still results in a noticeable gain, especially for the WENO scheme.



(a) 6th-order Central (b) 5th-order WENO

Figure 4: Effect of hyperthreading on weak scaling results for 6th-order central-difference (left) and 5th-order WENO (right) on the *Vulcan* BG/Q cluster, for a unit block size of 96^3 grid points per core, increasing the core-count from 32 to 131,072. Blue circles: no hyperthreading; green triangles: 2 MPI tasks per core; red squares: 4 MPI tasks per core. Top plots: average wall-clock time per time step divided by the number of grid points per core, t_w, in solid lines, and the contribution of MPI communications in dashed lines (same symbols). Bottom plots: weak scaling efficiency. Black dashed lines are drawn for reference to indicate the value achieved for the lower core-count and each specific configuration of numerical scheme and hyperthreading.

6.2 Strong scaling

Several strong scaling tests were performed. We present first results with a grid size of $8{,}192^3$ (approximately 550 billion) points, which is representative of the targeted scientific runs (see section 6.4). The memory footprint of the *Hybrid* code for this grid size is approximately 185 TB. For that memory usage, the starting point of this strong scaling study corresponds to 16 racks (262,144 cores), for which the memory usage per core for this grid size is approximately 707 MB. While keeping the problem size constant, we increase the number of cores in successive runs until reaching the full capacity of the *Sequoia* and *Vulcan* clusters combined

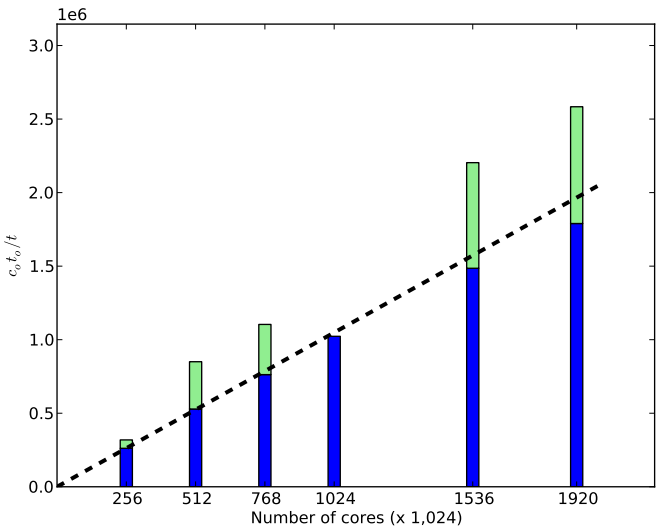
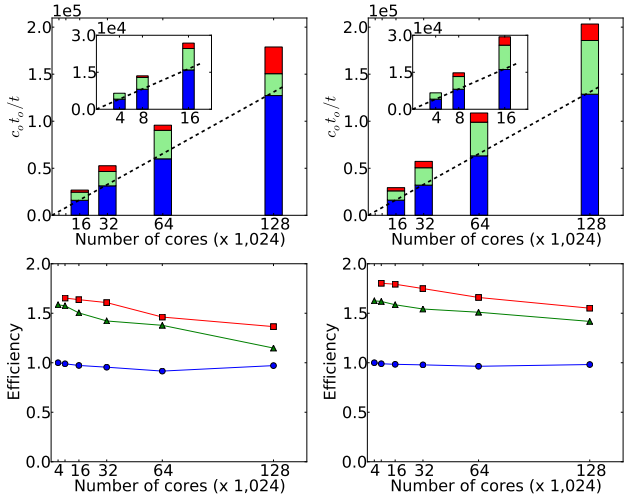


Figure 5: Strong scaling results for a problem size of 8,192^3 grid points, increasing the core-count from 262,144 to 1,966,080. Blue bars: 1 MPI task per core; green bars: 2 MPI tasks per core (data point not available at 1,048,576 cores). Dashed line: linear scaling for the case of one task per core.



(a) 6th-order Central (b) 5th-order WENO

Figure 6: Effect of hyperthreading on strong scaling results for 6th-order central-difference (left) and 5th-order WENO (right) on a problem size of 2,048^3 grid points, increasing the core-count from 4,096 to 131,072. Top plots show the speedup (the inset zooms in the lower core-count range); bottom plots show the strong scaling efficiency. Blue: 1 MPI task per core; green: 2 MPI tasks per node; red: 4 MPI tasks per node. Dashed black lines correspond to the linear scaling considering one task per core.

(1,966,080 cores). To evaluate the impact of hyperthreading in scalability, we perform runs first with 16 MPI tasks per node (i.e., one task per core), and also with 32 tasks per node (i.e., two per core). Only the central-difference scheme is used for this first strong-scaling test. The results are shown in figure 5, plotting parallel speedup vs number of cores. The dashed line represents the linear scaling starting from the case with the minimum number of cores and no hyperthreading. Substantial speedups are observed also in this strong-scaling study by means of hyperthreading, shown by the green bars, as the increment with respect to the base cases with no hyperthreading, shown in blue. No data could be retrieved for the 1,048,576 number of cores with hyperthreading during the time the clusters were available for this study.

To evaluate the behavior for smaller core counts, the grid size must be reduced. Thus, we present additional strong scaling studies considering a grid of 2,048^3 points, and increasing the number of cores from 4,096 to 131,072. In this study, we compare the WENO and central-difference numerical schemes, and also consider the two available levels of hyperthreading (2 and 4 MPI tasks per node). Figure 6 shows the results, in terms of speedup and parallel efficiency. The results are consistent with the previous comments for the other weak and strong scaling studies. Parallel efficiency is nearly linear for the WENO numerical scheme, as a result of an increased computational cost while keeping the communication time nearly identical to the central-difference scheme. The improvement provided by hyperthreading is slightly higher for the WENO scheme, and it can be better sustained as the number of cores increases, compared to the central-difference scheme. The largest gain in this range of core numbers is provided by using 4 MPI tasks per core, although the relative increase is consistently higher for 2 MPI tasks per node. We note that for the minimum core count considered in these runs (4,096), 4-way hyperthreading using MPI was not possible due to the memory footprint induced by MPI.

6.3 Performance

The objective of this section is to evaluate the hardware utilization efficiency of the *Hybrid* code in the BG/Q platform, for each numerical scheme and considering different grid sizes and levels of hyperthreading. The performance measurements were obtained using the IBM HPC Toolkit (see section 5) on the *Vulcan* cluster, with runs executed on two racks (32,768 cores), to involve inter-rack communications, and two grid sizes (2,048^3 and 3,072^3 points).

Table 2 summarizes the results. It can be first noticed that the higher algorithmic complexity of the 5th-order WENO scheme brings a larger average number of instructions issued per cycle per core in comparison with the 6th-order central-difference scheme which is purely stencil based. Independently from the grid size, both schemes take a large advantage of the hyperthreading: 2- and 4-way hyperthreading typically result in gain factors in the number of instructions issued per cycle per core of approximately 1.6 and 2, respectively, in comparison with cases in which a single hardware thread is used per core, for both numerical schemes. This is consistent with the ability of the hardware to issue two instructions per cycle instead of one, when more than one hardware thread is involved per core (see section 4).

As stated in section 4, of the two instructions that can be issued per cycle per core, one must come from each execution unit: XU issues integer/load/store operations whereas AXU issues floating-point operations. In all cases shown in

N	T	IPCPC		F		DTC		%P	
		C6	W5	C6	W5	C6	W5	C6	W5
2048^3	1	0.33	0.42	33	42	4.3	2.2	2.1	3.5
	2	0.52	0.67	28	40	6.8	3.9	2.9	5.3
	4	0.69	0.80	21	36	6.6	4.2	2.9	5.7
3072^3	1	0.32	0.41	34	43	4.8	2.4	2.2	3.5
	2	0.52	0.64	32	42	7.6	3.7	3.4	5.4
	4	0.65	0.81	28	40	8.1	4.4	3.6	6.4

Table 2: Performance results for 6th-order central-difference (C6) and 5th-order WENO (W5) schemes, on two racks (32,768 cores) of BG/Q. N: number of grid points; T: number of threads per core; IPCPC: instructions completed per cycle per core; F: percentage of floating-point instructions; DTC: DDR traffic (node average) (Bytes/cycle); %P: percentage of ideal peak performance.

table 2, the percentage of floating-point operations is lower than 50% of the total, as expected for this kind of memory-bound application. This percentage is always 9-15% higher for cases run with WENO vs central-difference, as a result of the more computationally intensive nature of WENO. A larger grid size, for a constant core-count, increases that percentage, whereas hyperthreading decreases it, more significantly for smaller grid sizes.

Regarding memory access, for the tests shown in table 2, cache hits are shared as follows: 88% goes to L1 while about 10% goes to the L2 cache. DDR traffic is relatively high for the central scheme with 8.1 bytes per cycle (the maximum bandwidth is approximately 18 bytes per cycle on Blue Gene/Q). It becomes much smaller (around 4) for the WENO scheme which is directly linked to an equivalent access in memory but a longer time to solution for this scheme. The same effect can be seen by the use of hyperthreading which increases the memory traffic. We note that little use of the L1 prefetch is found by the compiler (1.6% or less). Hardware counters confirm the high impact of L2 cache latency for our application, nevertheless greatly enhanced by the use of hyperthreading, which appears to be an efficient way to partially hide memory latency.

The previous measures combined lead to the global sustained level of performance, which is summarized in the rightmost column of table 2 for each run. Percentage refers to the ideal 419.4 TFLOPS that can be delivered by two BG/Q racks. WENO shows an overall performance 1.9 times higher, on average, than central-difference. For both numerical schemes, hyperthreading improves significantly the performance, particularly when the grid size of the problem is increased and therefore memory locality decreased. Changing from no hyperthreading to 2-way hyperthreading results in the largest performance gain, compared with the transition from 2- to 4-way hyperthreading. It is worth mentioning that for small grid sizes (not shown in the table), the overhead induced by MPI communications with 4-way hyperthreading leads to saturation of the benefits, bringing sometimes the performance to a lower level than that achieved with two hardware threads only.

The main bottlenecks in the execution of the solver were identified by profiling the *Hybrid* code on two racks of Vulcan, using each numerical scheme in two separate runs. We detail here the profiling results referred to the high-level description of the algorithmic steps given in section 3 and presented in Table 3. For the central-difference scheme, the interpolation required to obtain the inviscid fluxes took nearly 26% of the total time, to be added to 10% spent in the subsequent computation of the inviscid terms from those fluxes. The calculation of viscous fluxes accounts for approximately 12.5% of the total time for each one of the y and z coordinate directions, in contrast to only 9% spent to compute the viscous fluxes in x. An extra 8% of the time is dedicated to the additional operations required to compute the contribution from the viscous terms based on those calculated fluxes. The computation of the velocity gradients in y and z took slightly above 3.5% of the time, whereas the gradients in x required about 3%. This discrepancy among the execution times found for operations (e.g., gradients, viscous fluxes) in different coordinate directions is a direct consequence of the privileged data locality found for the x coordinate direction, which corresponds to the fastest-varying index in the arrays containing the data in memory for all the points within an MPI block. Setting up, initiating and finishing the non-blocking MPI halo-data exchange accounts for 7% of the total time, and the intrinsic array multiplications and additions in the Runge-Kutta time-stepping update take less than 5% of the execution. In contrast, when the WENO scheme is used, the interpolation required to obtain inviscid fluxes, together with the Roe flux splitting, dominates the profiling results, with about 60% of the time spent in it. The other elements described above when using central-difference scheme remain in practically the same order, with their percentages of total time significantly brought down due to a smaller relative contribution.

	C6 (%)	W5 (%)
Convection	**36** (26+10)	60
Diffusion	**42** (12.5+12.5+9+8)	26
Gradients computation	**10** (3.5 + 3.5 +3)	6
Data re-arranging	**7**	5
Runge-Kutta update	**5**	3

Table 3: Relative timing in advancement of one time step. Addition between brackets for the central scheme refers to the substep discussed in the text.

6.4 Time-to-solution

The state-of-the-art in direct numerical simulation (DNS) of shock/turbulence interaction is represented by the blue line in figure 2, which corresponds to the simulations on $2,366 \times 1,024^2$ grid points ran for about 35,000 time steps on 65,536 cores of the Blue Gene/P *Intrepid* machine during approximately 17 wall hours [9]. As discussed in section 1, there is great scientific value in reaching a Reynolds number high enough to have a "spectral gap" (i.e., a non-overlapping range) between the energetic and dissipative length scales of motion. From figure 2, this suggests that the ratio L/η must be increased by a factor of about 4 above the prior state-of-the-art (corresponding to a Taylor microscale of the incoming turbulence before meeting the shock of $Re_\lambda \approx 190$). This simulation of shock/turbulence interaction would require about $9{,}464 \times 4{,}096^2 = 160 \cdot 10^9$ grid points and 140,000 time steps.

The time-to-solution for such simulation on the full Sequoia cluster can be estimated considering the strong scaling study reported in figure 5, which required 5.71 wall-

clock seconds per time step on 1,572,864 cores with 2-way hyperthreading. This implies a normalized $t_w = 16$ μs. This case used the central-difference scheme, and thus the shock/turbulence case would be slightly more expensive due to the solution-adaptive algorithm required to combine central-difference and WENO (applied across the shock wave). The exact factor depends on how close to optimal load-balance can be achieved, but our prior experience suggests that the factor is often around 1.5 and rarely larger than 2. Taking the worst case scenario then gives an estimated total time of $2 \times 140{,}000 \times (9{,}464 \times 4{,}096^2) \times 16 \cdot 10^{-6}/3600 \approx 200 \cdot 10^6$ core hours, which would take about 126 wall hours (≈5 days) on the full *Sequoia* cluster.

Doubling the grid resolution to a number of $18,928 \times 8,192 \times 8,192 = 1.27$ trillion points, with a corresponding $Re_\lambda \approx 300$ of the incoming turbulence, would still allow the use of hyperthreading, carrying an associated cost of $3.16 \cdot 10^9$ core hours, feasible in about 84 days of full utilization of Sequoia.

7. CONCLUSIONS

The performance of the *Hybrid* code, a finite-difference solver for high-fidelity simulations of compressible turbulence and shock waves was assessed on the IBM Blue Gene/Q *Sequoia* and *Vulcan* clusters at the Lawrence Livermore National Laboratory during an early-access period prior to Sequoia becoming classified. The main focus of the study was to assess the scaling efficiency of the code up to the full 1.97 million cores made available by the temporary connection of the *Sequoia* (1.6 million cores) and *Vulcan* (393,216 cores) clusters.

The code was found to scale extremely well, in both strong and weak scaling studies. The explicit nature of the code, in both space and time, is a main reason for this; in addition, the code makes use of non-blocking MPI communication, with concurrent computation during the communication phase. The performance and scalability of the two numerical schemes (6th-order central-difference and 5th-order WENO) involved in the simulation of shock-turbulence interaction with *Hybrid* were evaluated individually to provide estimates that can be used for an appropriate load-balanced decomposition of future scientific runs.

The code has a low memory footprint and does not require large shared memory usage at the node level. Therefore, hyperthreading is accessible through MPI directly, even though memory overhead induced by MPI is not negligible when using $O(10^6)$ tasks. Hyperthreading was found to increase the performance by at least 50% and as much as 80% for the case of the (computationally more intensive) WENO scheme with 4-way hyperthreading. The low memory usage led to the possibility to perform a simulation with $15{,}360 \times 16{,}384 \times 16{,}384 = 4.12 \cdot 10^{12}$ grid points, using nearly the full memory capacity of the combined *Sequoia*/ *Vulcan* machine.

Future phases of this work will focus on two goals. From a physics standpoint, we will target production simulations of shock-turbulence interaction at higher Reynolds numbers, for which a spectral gap is present between energetic and dissipation scales. From a high-performance computing standpoint, we will evaluate modifications of the data structures of the *Hybrid* code to optimize data locality, thus minimizing memory traffic and improving performance.

8. ACKNOWLEDGMENTS

The authors are grateful to Bob Walkup from the IBM Thomas J. Watson Research Center (Advanced Computing Technology Center) for valuable discussions on high-performance computing, to John Gyllenhaal, Scott Futral, David Fox and Richard Hedges from the Lawrence Livermore National Laboratory for their technical support and advise in the use of the *Sequoia* and *Vulcan* clusters, and to Ramesh Balakrishnan and Vitali Morozov at the Argonne Leadership Computing Facility (Argonne National Laboratory) for providing access to the *Vesta* cluster and for their help with code optimization.

The *Sequoia* and *Vulcan* IBM BG/Q systems at (LLNL) are funded by the Advanced Simulation and Computing (ASC) Program of the National Nuclear Security Administration (NNSA). Access to *Sequoia* and *Vulcan* clusters was provided through the Predictive Science Academic Alliance Program (PSAAP). Access to *Vesta* was provided through the AURORA discretionary program. The *Hybrid* code was originally written by Johan Larsson when he was a Postdoctoral Fellow and Research Associate at the Center for Turbulence Research under the SciDAC program. Financial support from the Department of Energy is acknowledged.

9. REFERENCES

[1] N. A. Adams and K. Shariff. A high-resolution hybrid compact-ENO scheme for shock-turbulence interaction problems. *J. Comput. Phys.*, 127:27–51, 1996.

[2] I.-H. C. an C. Kim, H.-F. Wen, and G. Cong. Application Data Prefetching on the IBM Blue Gene/Q Supercomputer. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing, SC12*, pages 1–8, 2012.

[3] D. A. Donzis, P. K. Yeung, and D. Pekurovsky. Turbulence Simulations on $O(10^4)$ Processors. In *TeraGrid 08*, June 2008.

[4] M. Gilge. *IBM System Blue Gene Solution Blue Gene/Q Application Development*. International Business Machines Coorporation, 2013.

[5] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Chist, and C. Kim. The IBM Blue Gene/Q compute chip. In *IEEE Computer Society*, volume 32, pages 48–60, 2012.

[6] B. Hejazialhosseini, D. Rossinelli, C. Conti, and P. Koumoutsakos. High throughput software for direct numerical simulations of compressible two-phase flows. In *Proceedings of the 2012 ACM/IEEE Conference on High Performance Computing SC12*, Nov. 2012.

[7] T. Ishihara, T. Gotoh, and Y. Kaneda. Study of High-Reynolds Number Isotropic Turbulence by Direct Numerical Simulation. *Annu. Rev. Fluid Mech.*, 41(1):165–180, 2009.

[8] E. Johnsen, J. Larsson, A. V. Bhagatwala, W. H. Cabot, P. Moin, B. J. Olsson, P. S. Rawat, S. K. Shankar, B. Sjögreen, H. C. Yee, X. Zhong, and S. K. Lele. Assessment of high-resolution methods for numerical simulations of compressible turbulence with shock waves. *J. Comput. Physics*, 229:1213–1237, 2010.

[9] J. Larsson, I. Bermejo-Moreno, and S. K. Lele. Reynolds- and Mach-number effects in canonical shock-turbulence interaction. *J. Fluid Mech.*, 717:293–321, 2013.

[10] J. Larsson and S. K. Lele. Direct numerical simulation of canonical shock/turbulence interaction. *Phys. Fluids*, 21:126101, 2009.

[11] S. Lee, S. K. Lele, and P. Moin. Interaction of isotropic turbulence with shock waves: effect of shock strength. *J. Fluid Mech.*, 340:225–247, 1997.

[12] K. Mahesh, S. K. Lele, and P. Moin. The influence of entropy fluctuations on the interaction of turbulence with a shock wave. *J. Fluid Mech.*, 334:353–379, 1997.

[13] C. Pantano, R. Deiterding, D. J. Hill, and D. I. Pullin. A low numerical dissipation patch-based adaptive mesh refinement method for large-eddy simulation of compressible flows. *J. Comput. Phys.*, 221:63–87, 2007.

[14] D. Pekurovsky. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. *SIAM J. Sci. Comput.*, 34(4):C192–C209, 2012.

[15] S. Pirozzoli. Conservative hybrid compact-WENO schemes for shock-turbulence interaction. *J. Comput. Phys.*, 178:81–117, 2002.

[16] S. Pirozzoli and M. Bernardini. Probing high-Reynolds-number effects in numerical boundary layers. *Phys. of Fluids*, 25(2):021704:1–7, 2013.

[17] R. S. Rogallo. Numerical experiments in homogeneous turbulence. TM 81315, NASA, 1981.

[18] P. Schlatter and R. Orlu. Assessment of direct numerical simulation data of turbulent boundary layers. *J. Fluid Mech.*, 659:116–126, 2010.

[19] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.